Lesson 6

- Overfitting
- Entropy, CrossEntropy, Kullabck-Leibler divergence

Overfitting

Overfitting and underfitting

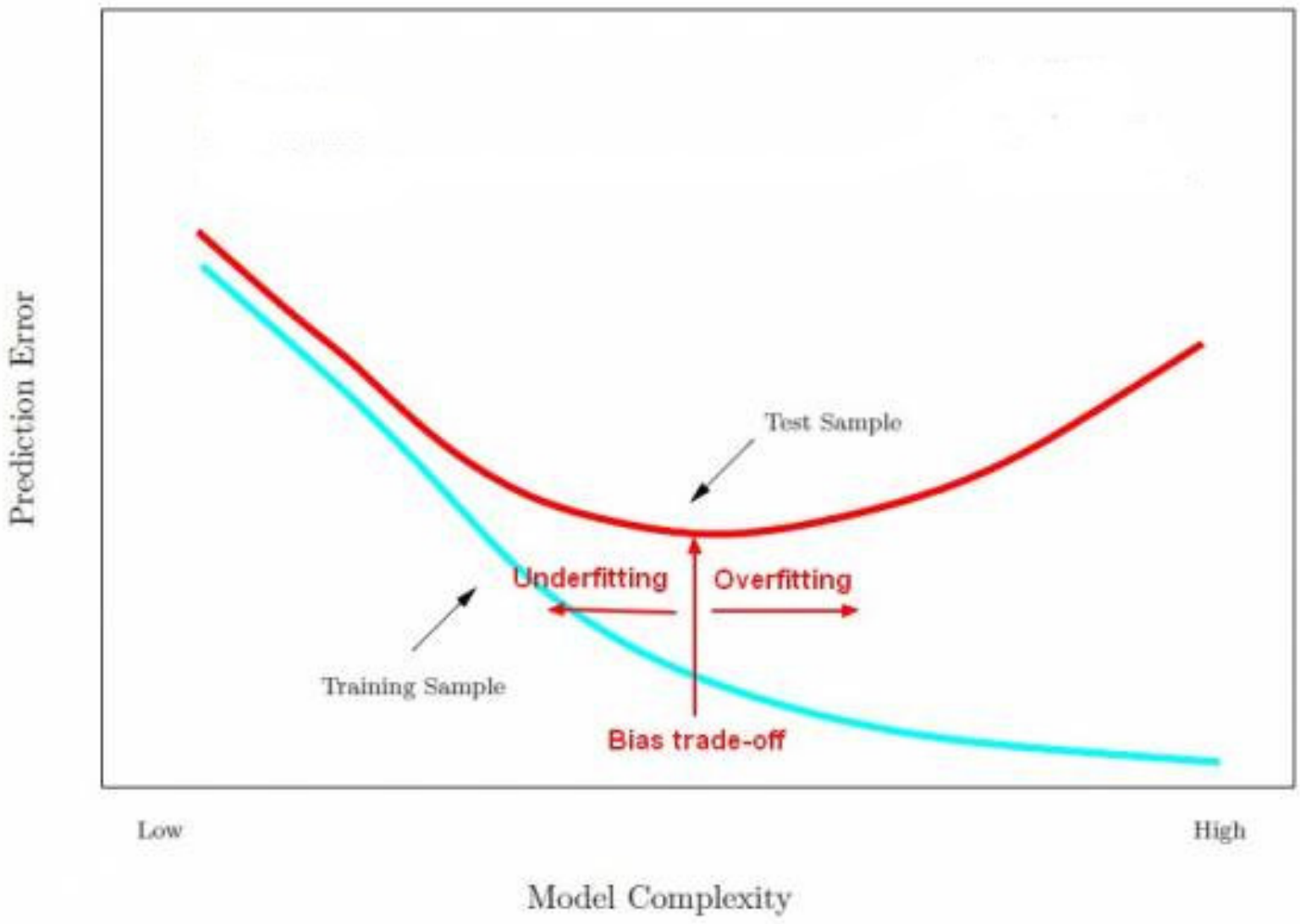
overfitting the model is too complex and specialized over the peculiarities of the samples in the training set

underfitting the model is too simple and does not allow to express the complexity of the observations.

remark

Deep models are good at fitting, but the real goal is generalization

Overfitting and model complexity



About overfitting

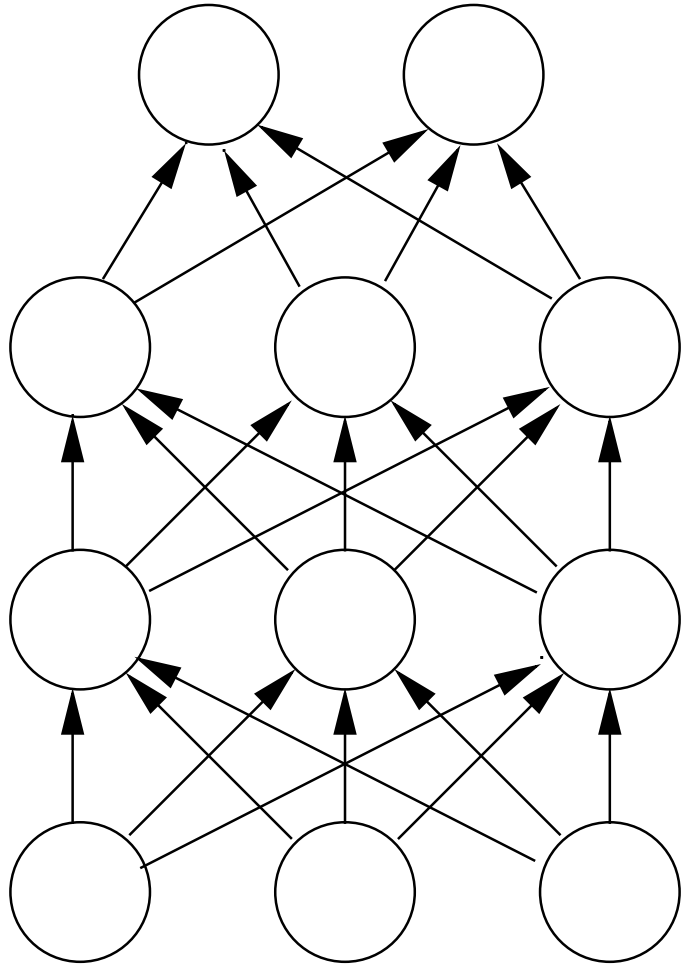
Ways to reduce overfitting

- Collect more data
- Reduce the model capacity → simplify the network capacity
- Early stopping → monitor the model's performance and stop when not good
- Regularization, e.g. Weight-decay → penalize large weights and complex models
- Model averaging → combine predictions of different instances
- Data augmentation → introduce variations to training set
- Dropout → during learning we randomly shut down some neurons in each layer, preventing too many specific connections

Dropout

Idea: "cripple" the neural network stocastically removing hidden units

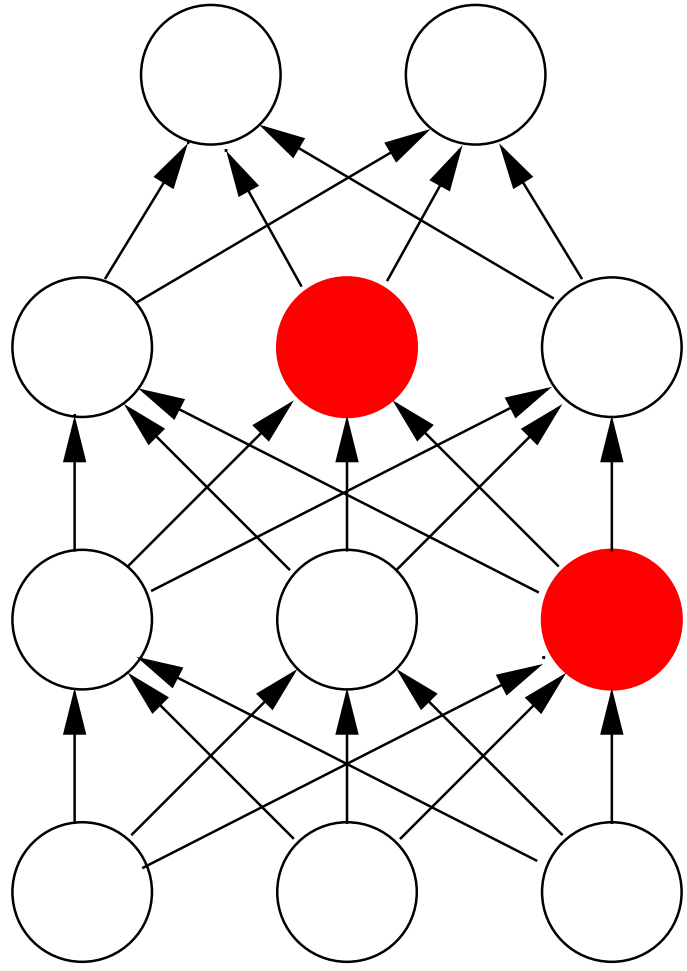
- during training, at each iteration hidden units are disabled with probability p (e.g. 0.5)
- hidden units cannot co-adapts with other units
- similar to train many networks and averaging between them



Dropout

Idea: "cripple" the neural network stocastically removing hidden units

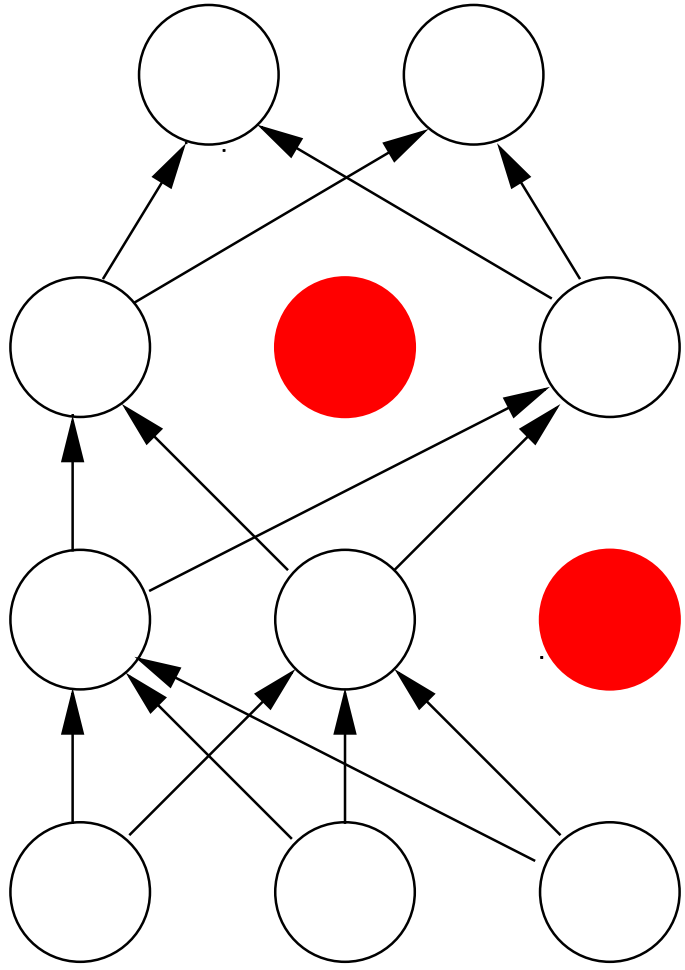
- during training, at each iteration hidden units are disabled with probability p (e.g. 0.5)
- hidden units cannot co-adapts with other units
- similar to train many networks and averaging between them



Dropout

Idea: "cripple" the neural network stocastically removing hidden units

- during training, at each iteration hidden units are disabled with probability 1-p (e.g. 0.5)
- hidden units cannot co-adapts with other units
- similar to train many networks and averaging between them



Geometric averaging

At each stage of training, only the crippled network is trained by means of backpropagation. Then, the omitted units are reinserted and the process repeated (hence weights are shared among the crippled networks).

At test time, we weight each unit with its expectaction p.

For a single layer, this is equivalent to take a geometric average among all different crippled networks.

A form of regularization

With Dropout, we are randomly sampling from an exponential number of different architectures
- all architectures share weights

Sharing weights means that every model is very strongly regularized, by all the other models
- A good alternative to L2 or L1 penalties that pull the weights towards zero.

Demonstrating Overfitting

Example 1: The IMDB Movie reviews data set

A Dataset of 25,000 movies reviews from IMDB, labeled by sentiment (positive/negative).

Each review is a sequence of words in a vocabulary of 10000 different words. Each word is encoded by an index (integer) in the range [0,9999].

Words are indexed by overall frequency in the dataset; for instance, the integer "3" encodes the 3rd most frequent word in the data.

This allows for quick filtering operations such as: "only consider the top 10,000 most common words, but eliminate the top 20 most common words".

Bag of words approach

Encode each review r as a boolean vector x_r of dimension 10000 (number of different words).

We neglect the order and the multiplicity.

$x_r[i] = 1$ if the word with index i appears in the review r, and 0 otherwise.

DEMO

Example 2: CIFAR-10

DEMO
(data augmentation)

Suggested reading:

Do CIFAR-10 Classifiers Generalize to CIFAR-10?

Activation and loss functions for classification

Sigmoid

When the result of the network is a value between 0 and 1, e.g. a probability for a binary classification problem, it is customary to use the sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}$$

as activation funtion.
If

$$P(Y=1|x) = \sigma(f(x)) = \frac{e^{f(x)}}{1+e^{f(x)}}$$

then

$$P(Y=0|x) = 1-\sigma(f(x)) = \frac{1}{1+e^{f(x)}}$$

Softmax

When the result of the network is a probability distribution, e.g. over K different categories, the softmax function is used as activation:

$$\text{softmax}(j, x_1, \ldots, x_k) = \frac{e^{x_j}}{\sum_{j=1}^{k} e^{x_j}}$$

It is easy to see that

$$0 < \text{softmax}(j, x_1, \ldots, x_k) < 1$$

and most importantly

$$\sum_{j=1}^{k} \text{softmax}(j, x_1, \ldots, x_k) = 1$$

since we expect probablilites to sum up 1.

Softmax vs Sigmoid

It is easy to prove that for any c,

$$\text{softmax}(j, x_1 + c, \ldots, x_k + c) = \text{softmax}(j, x_1, \ldots, x_k)$$

in particular, we can always assume one argument (corresponding to a "reference category") is null, taking e.g. $c = -x_k$.

In the binary case, we would be left with a single argument, and in particular

$$\sigma(x) = \text{softmax}(x, 0) = \frac{e^x}{e^x + e^0} = \frac{e^x}{e^x + 1}$$

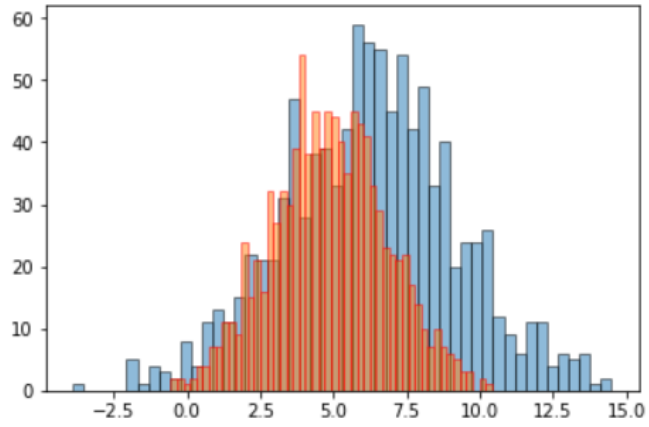
Cross entropy

Loss functions for probability distributions

If the intended output of the network is a probability distribution, we should find ways to compare it with the ground truth distribution (usually, but not necessarily, a categorical distribution).

Loss functions

What loss functions should we use for comparing probability distributions?



We could treat them as "normal functions", and use e.g. quadratic distance between true and predicted probabilities.

Can we do better? For instance, in logistic regression we do not use mean squared error, but use negative loglikelihood. Why?

GO TO

SLIDE 6_ MODIFIED